Extracting Decision Logic from Process Models

Kimon Batoulis[1(✉)], Andreas Meyer[1], Ekaterina Bazhenova[1], Gero Decker[2], and Mathias Weske[1]

[1] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{kimon.batoulis,andreas.meyer,ekaterina.bazhenova,mathias.weske}@hpi.de
[2] Signavio GmbH, Berlin, Germany
gero.decker@signavio.com

Abstract. Although it is not considered good practice, many process models from practice contain detailed decision logic, encoded through control flow structures. This often results in spaghetti-like and complex process models and reduces maintainability of the models. In this context, the OMG proposes to use the Decision Model and Notation (DMN) in combination with BPMN in order to reach a separation of concerns. This paper introduces a semi-automatic approach to (i) identify decision logic in process models, (ii) to derive a corresponding DMN model and to adapt the original process model by replacing the decision logic accordingly, and (iii) to allow final configurations of this result during post-processing. This approach enables business organizations to migrate already existing BPMN models. We evaluate this approach by implementation, semantic comparison of the decision taking process before and after approach application, and an empirical analysis of industry process models.

Keywords: Process modeling · Decision modeling · BPMN · DMN

1 Introduction

Business process models are important artifacts in today's business organizations, since they provide expressive means to represent business logic. The corner stones of business process models are work activities, their logical ordering, data, and organizational responsibilities. With these models, organizations can improve, control, automatize, and measure their processes effectively [15]. In our studies of business process models from our project partners, we have also found situations, in which business process models were misused for modeling decision logic. The respective process models expose a complex routing structure, consisting of many exclusive gateways that represent different aspects of a decision. As a result, these process models are hard to comprehend, to implement, and to maintain.

In this paper, we argue that decision logic should be modeled separately from the process logic. Following the "separation of concerns" paradigm, this allows to keep the decision logic in a dedicated decision model and the process logic in a dedicated process model. To take advantage from existing information,

J. Zdravkovic et al. (Eds.): CAiSE 2015, LNCS 9097, pp. 349–366, 2015.
DOI: 10.1007/978-3-319-19069-3_22

we introduce an approach to semi-automatically detect decision logic in business process models and to generate decision models and corresponding decision tables from process models' decision logic. These decision models conform to the recently published Decision Model and Notation (DMN) [9] standard for decision modeling. Process models are represented with the industry standard: the Business Process Model and Notation (BPMN) [8]. The conceptual results are evaluated by sets of sample business processes from industry.

The paper contains two main contributions shaping its structure. First, we discuss the need to separate process and decision modeling (Section 2). Second, we introduce a semi-automatic approach to identify decision logic in business processes pattern-based (Section 3), to map these patterns into DMN models and to adapt the process model structure accordingly (Section 4) before we allow configuration of the results in post-processing (Section 5). Afterwards, Section 6 evaluates our approach and introduces our implementation. Section 7 is devoted to related work and Section 8 concludes the paper.

2 Process and Decision Modeling

Business process modeling is well established in business organizations and is highly supported by modern tools. Modeling languages as BPMN [8] are well suited for process experts as well as – thanks to tool support – end users. Process models allow documentation of business operations, enactment of business processes following the given execution semantics, and (automatic) process analysis regarding correctness, improvement, compliance, etc. In the course of process execution, multiple decisions are taken that influence the mentioned areas [2]. Analysis of industry processes reveals that such decisions include the assignment of actual resources to activities answering the question who shall execute a specific activity or evaluating a given set of data to calculate a decision indicating which path shall be followed at decision points (branching behavior). Furthermore, exceptional behavior is handled by pre-specified procedures for each case. Especially, in the insurance and banking domains, regulatory compliance highly influences process execution by specifying which guidelines must be followed.

Based on the analysis of 956 real world process models from, amongst others, insurance, banking, and health care, we recognized that part of the logic leading to decisions is often encoded in process models resulting in models that are hard to read and to maintain. BPMN allows to represent decisions and their impact or consequence respectively. However, BPMN is not meant to represent the detailed decision logic since modeling the decision logic often results in spaghetti like models (see Fig. 1 for an abstract example) or extensive natural language descriptions explaining the decision

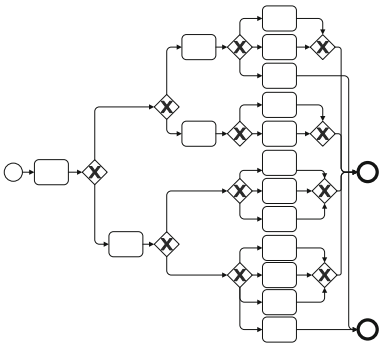


Fig. 1. Misuse of BPMN for decision logic modeling

taking process that are not analyzable automatically. Thus, decision logic modeling is out of scope for BPMN. Instead, decision tables [14] and further decision modeling concepts [7,16] are a reasonable and compact method to describe decision logic in scenarios with many input parameters. There exists the upcoming OMG standard for modeling decision diagrams: the Decision Model and Notation (DMN) [9]. DMN is meant to supplement BPMN and allows the "separation of concerns" [10] between process and decision logic modeling based on the actual scope of both modeling techniques. BPMN's scope comprises the business logic containing information on what activities need to be executed in which order by which resource utilizing which data objects while DMN covers the decision logic modeling by specifying which decision is taken based on which information, where to find it, and how to process this information to derive the decision result. Since both worlds existed long without proper integration, organizations misused BPMN by including decision logic into process models.

Thereby, data-based decisions are most common. A data-based decision is represented by a decision structure consisting of single and compound decision nodes we refer to as split gateways representing exclusive or inclusive alternatives based on external information. These decisions can be classified into three types: (i) An explicit decision task with succeeding branching behavior, e.g., in a task, the decision about a customer's loyalty is taken and based on the result, different actions (e.g., give or deny discount) are taken. (ii) Branching behavior is encoded in decision points, e.g., split gateways with decision logic (about the customer's loyalty) encoded in annotations to the gateways or to edges originating from such gateway. (iii) There exists a decision task without succeeding branching behavior, e.g., set discount for a customer based on her loyalty.

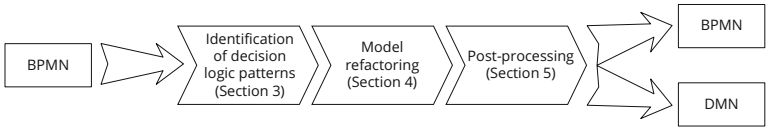


Fig. 2. Semi-automatic three step approach for process and decision logic separation

Fig. 2 visualizes the three main steps of our approach and the corresponding input and output. Given a BPMN model, we first identify decision logic patterns based on the specifications of Section 3. From the industry process model collections, we identified the three most often occurring patterns. We consider control-flow based decision structures only. Based on our insights into the industry process models, we defined three decision structures based on the occurrence of activities and gateways. However, we allow utilization of such information in the refactoring or the post-processing steps which follow in this order upon identification completion. The identification step is completed if a stakeholder approved the found patterns to be decision structures. Thereby, multiple patterns may match for parts of the process model such that the stakeholder also

must decide which is the appropriate pattern. The refactoring is presented in Section 4 and comprises the translation of the identified patterns into a DMN model and the adaptation of the process model. The post-processing step, see Section 5, enables configuration of the resulting BPMN and DMN model. Two configuration options and their automatic application are discussed in the corresponding section. Finally, both models, the BPMN process model and DMN decision model are the outputs of our semi-automatic approach.

3 Patterns for Control-Flow-Based Decision Identification

We analyzed seven industry process model collections from the domains of insurance, banking, healthcare, energy, and information technology with each collection containing between 14 and 334 process models summing up to 956 process models in total. From these process models, the majority, 63%, contain data-based decisions. Note, that some process models contain multiple types of decisions. So, in total we observed 1,074 decision occurrences. Following the empirical results, in this paper, we focus on process models with data-based decisions that are taken within tasks directly preceding a split gateway where this gateway only routes the process flow based on the taken decision: branching behavior with explicit decision task. Fig. 3 presents such decision structure consisting of a single gateway in an insurance environment.

Each decision structure is a fragment of the process model the decision is taken in. We formally define the concepts of process model and process fragment as follows.

Definition 1 (Process model). *Process model* $m = (N, D, \Sigma, C, F, \alpha, \xi)$ consists of a finite non-empty set N of control flow nodes, a finite set D of data nodes, a finite set Σ of conditions, a finite set C of directed control flow edges, and a finite set F of directed data flow edges. The set of control flow nodes $N = A \cup E \cup G$ consists of mutually disjoint sets $A \subseteq T \cup S$ of activities being tasks T or subprocesses S, set E of events, and set G of gateways. $C \subseteq N \times N$ is the control flow relation such that each edge connects two control flow nodes. $F \subseteq (D \times A) \cup (A \times D)$ is the data flow relation indicating read respectively write operations of an activity with respect to a data node. Let Z be a set of control flow constructs. Function $\alpha : G \to Z$ assigns to each gateway a type in terms of a control flow construct. Function $\xi : (G \times N) \cap C \nrightarrow \Sigma$ assigns conditions to control flow edges originating from gateways with multiple outgoing edges. $\diamond$

Definition 2 (Process fragment). Let $m = (N, D, \Sigma, C, F, \alpha, \xi)$ be a process model. A *process fragment* $pf = (N', D', \Sigma', C', F', \gamma, \sigma)$ is a connected subgraph of process model m such that $N' \subseteq N$, $D' \subseteq D$, $\Sigma' \subseteq \Sigma$, $C' \subseteq C$, and $F' \subseteq F$. Functions γ and σ are restrictions of functions α and ξ respectively with corresponding new domains. $\diamond$

We use subscripts, e.g., A_m, α_m, and N_{pf}, to denote the relation of sets and functions to process model m or process fragment pf and omit the subscripts

where the context is clear. The set of process fragments of process model m is denoted as PF_m. In this paper, we consider XOR and IOR as possible control flow constructs referring to an exclusive choice and an inclusive choice respectively. An XOR or IOR gateway with two or more outgoing edges is called split and an XOR or IOR gateway with two or more incoming edges is called join, whereby we assume that such gateway is either a split or a join. A gateway with multiple incoming and outgoing edges is transformed in two succeeding gateways with one representing the join and the other the split in this order.

As usual, we assume the process model to be structurally sound, i.e., m contains exactly one start and one end event and every node of m is on a path from the start to the end event. Further, we require each process fragment pf to consist of a single start node being an activity, multiple end nodes being activities (one for each alternative), each node is on a path from the start to some end node, and all nodes must not be a join gateway. We assume that decisions are taken in distinct tasks such that a split gateway only routes the process flow based on pre-calculated decision values, since it is good modeling practice to do so. "Omission of this decision task is a common mistake made by inexperienced BPMN users" [14].

We chose the top-three patterns in terms of occurrences in the industry process models as most prominent examples to show feasibility of combining BPMN and DMN automatically. A generalization of these patterns is out of scope for this paper due to the diverse nature of decision modeling. Next, we introduce these three identified patterns. Each pattern is represented as process fragment. Thereby, we utilize a fragment of a process model from the insurance domain dealing with assigning the correct discount for a customer. In the examples, for clarity reasons, we sometimes visualize two outgoing edges only for a split gateway. However, in practice, there can be any number of edges as covered by the corresponding formalisms.

3.1 P1 – Single Split Gateway

A fragment matching pattern P1 contains a decision structure of a task preceding a single split gateway with at least two outgoing control flow edges. On each path, an activity directly succeeds the split gateway. Thereby, pattern P1 subsumes optionality decisions as special case; paths directly connecting split and join gateways get automatically extended by τ-transitions. We assume, the decision is taken in the task preceding the gateway. Fig. 3 presents a corresponding process fragment with three alternative paths at the split gateway; depending on the modeling style, the bottom activity *assign no discount* might not have been modeled such that it would have been added as τ-transition instead. Since, the gateway is of type XOR, only one alternative can be chosen. Based on the result of task

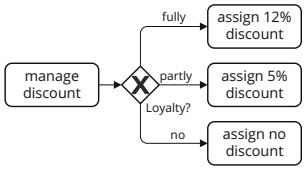


Fig. 3. Process fragment representing a split gateway with more than 2 outgoing edges

manage discount, i.e., the taken decision about the customer's loyalty, the discount assigned to the customer is set to 12%, 5%, or 0% respectively. Possible results of the decision are fixed by the annotations on the edges originating from the split gateway.

Formally, we specify pattern P1 as follows.

Definition 3 (Pattern P1). Let pf be a process fragment of process model m and let Σ_{pf} denote the conditions assigned to control flow edges. Then, pf represents *P1* if

- $|G_{pf}| = 1 \wedge |g\bullet| \geq 2 \wedge (\gamma(g) = XOR \vee \gamma(g) = IOR)$, $g \in G_{pf}$ (the fragment contains exactly one split gateway),
- $|A_{pf}| = |g\bullet|+1$ (the number of activities of pf equals the number of outgoing edges[1] of the split gateway g plus 1),
- $\bullet g = t \wedge |\bullet g| = 1$, $t \in T_{pf}$ (task t is the only predecessor of the split gateway),
- $|\bullet t| = 0$ (task t is the start node of pf),
- $\forall a \in A_{pf} \backslash t : \bullet a = g$ (all activities other than the one preceding the split gateway g directly succeed g),
- $\forall a \in A_{pf} \backslash t : |a\bullet| = 0$ (all activities other than the one preceding the split gateway g are end nodes of pf), and
- $\forall a \in A_{pf} \backslash t, c \in C_{pf}$ such that $(g, a) = c : \sigma(c) \in \Sigma_{pf}$ (all outgoing edges of the split gateway are annotated with a condition). ◇

3.2 P2 – Sequence of Split Gateways (Decision Tree)

A fragment matching pattern P2 contains a decision structure of a task preceding a split gateway with at least two outgoing control flow edges. On each path, an activity or another split gateway with at least two outgoing control flow edges directly succeeds the split gateway. In case of a gateway, this proceeds iteratively until all paths reach an activity; i.e., on each path from the first split gateway to some end node of the fragment, there exists exactly one activity – the end node. We assume, all decisions are taken in the task preceding the first split gateway. Fig. 4 presents a corresponding process fragment with altogether four alternative paths after the first split gateway. Since, all gateways are of type XOR, only one alternative can be chosen. The actual routing based on the taken decisions is distributed over two split gateways. Based on the result for the customer loyalty, the second routing decision is either taken based on the longevity of the customer relationship (loyal customer) or the age of the customer (non-loyal customer). Due

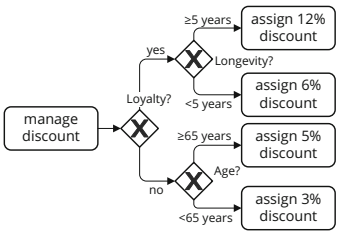


Fig. 4. Process fragment representing a sequence of split gateways that represents a decision tree

[1] The number of outgoing (incoming) edges directly translates to the number of direct successors (predecessors) and vice versa.

to the dependency of a routing decision on the ones taken before, this pattern represents a decision tree. Analogous to pattern P1, the possible results of the decision are fixed by the annotations on the edges originating from some split gateway.

Formally, we specify pattern P2 as follows.

Definition 4 (Pattern P2). Let pf be a process fragment of process model m and let Σ_{pf} denote the conditions assigned to control flow edges. Then, pf represents *P2* if

- $\forall g \in G_{pf} : |g \bullet| \geq 2 \wedge (\gamma(g) = XOR \vee \gamma(g) = IOR)$ (all gateways of the fragment are split gateways),
- $\exists t \in A_{pf} : |\bullet t| = 0 \wedge \forall a \in A_{pf} \backslash t : \bullet a = 1$ (t is the start node of pf),
- $\forall a \in A_{pf} \backslash t : |a \bullet| = 0$ (activities other than the start node t are end nodes of pf),
- $\forall g \in G_{pf} : \forall n \in g\bullet : n \in A_{pf} \cup G_{pf}$ (all successors of a gateway are an activity or a gateway), and
- $\forall a \in A_{pf} \backslash t, g \in G_{pf}, c \in C_{pf}$ such that $(g, a) = c \vee (g, g) = c : \sigma(c) \in \Sigma_{pf}$ (all outgoing edges of a split gateway are annotated with a condition).

◇

3.3 P3 – Sequence of Split Gateways Separated by an Activity

A fragment matching pattern P3 contains a decision structure of a task preceding a split gateway with at least two outgoing control flow edges. On each path, an activity or another split gateway with at least two outgoing control flow edges directly succeeds the split gateway. A task – a specific type of activity – that succeeds a split gateway may be succeeded by another split gateway. Otherwise, it is an end node of the process fragment. Activities of type subprocess are also end nodes of the fragment. Iteratively, this proceeds until all paths reach an activity that is not succeeded by some split gateway. Fig. 5 presents a corresponding process fragment with altogether three alternative paths after the first split gateway. Since, all gateways are of type XOR, only one alternative can be chosen. Each task of this process fragment that is succeeded by a split gateway (tasks *manage discount* and *check longevity* in Fig. 5) takes the actual decisions for the subsequent routing decisions. In case there exist multiple split gateways (see decision tree in pattern P2), the task takes the decisions for the whole decision tree. This means, this pattern can be composed of multiple decision trees as well as single split gateways. Since multiple decisions are arranged in sequence, we consider this structure as additional pattern to preserve the decision dependencies instead of handling each decision separately. In Fig. 5, the choice between 12%

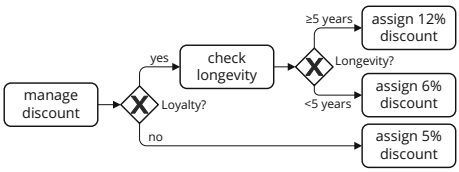


Fig. 5. Process fragment representing a sequence of split gateways separated by an activity

and 6% discount is taken based on two decisions (loyalty and longevity) while granting 5% discount is clear after the first decision for non-loyal customers.

Formally, we specify pattern P3 as follows.

Definition 5 (Pattern P3). Let pf be a process fragment of process model m and let Σ_{pf} denote the conditions assigned to control flow edges. Then, pf represents *P3* if

- $\forall g \in G_{pf} : |g\bullet| \geq 2 \wedge (\gamma(g) = XOR \vee \gamma(g) = IOR)$ (all gateways of the fragment are split gateways),
- $\exists t \in A_{pf} : |\bullet t| = 0 \wedge \forall a \in A_{pf} \backslash t : \bullet a = 1$ (t is the start node of pf),
- $\forall a \in A_{pf}$ such that $|a\bullet| = 1 : a \in T_{pf}$ (all activities being no end node are a task),
- $\forall g \in G_{pf} : \forall n \in g\bullet : n \in A_{pf} \cup G_{pf}$ (all successors of a gateway are an activity or a gateway),
- $\forall n \in N_{pf}$ such that $|n\bullet| = 0 : n \in A_{pf}$ (all end nodes are activities),
- $\forall a \in A_{pf} \backslash t, g \in G_{pf}, c \in C_{pf}$ such that $(g, a) = c \vee (g, g) = c : \sigma(c) \in \Sigma_{pf}$ (all outgoing edges of a split gateway are annotated with a condition).

◇

3.4 Pattern Identification Procedure

Given a process model, we check for the existence of decision logic by following the steps as shown in Fig. 6. For pattern identification, we first determine for all pairs of directly succeeding control flow nodes where the first one is a task, the decision task, and the second one is a split gateway. For each such pair of nodes, we traverse forward the process model and check for existence of a control flow structure aligning to the patterns defined above.

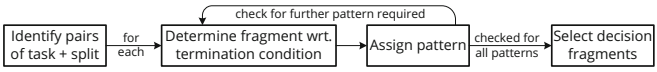


Fig. 6. Visualization of pattern identification process

For pattern P1, we check whether each path originating from the split gateway proceeds with an activity; such fragment is referred to pattern P1. Otherwise, P1-identification is stopped. For pattern P2, we traverse forward on each path until we identify a non-split-gateway control flow node, e.g., an activity or a join gateway, directly succeeding some split gateway. The initial split gateway must be followed by at least one other split gateway. Otherwise, P2-identification is stopped. For pattern P3, we traverse forward on each path until we identify a task that is directly succeeded by some control flow node that is no split gateway or until we identify a subprocess directly succeeding a split gateway. In case a split gateway is not succeeded by an activity or another split gateway or if a task is not succeeded by a split gateway, P3-identification is stopped.

After fragment determination, the fragment is referred to the pattern it was checked for if it was not stopped. Otherwise, the assignment is skipped. After checking each determined pair, the stakeholder gets presented all fragments that refer to some pattern and is required to decide which actually represent a decision. Thereby, process fragments indicating some decision may overlap. This needs to be resolved by the stakeholder resulting in non-overlapping fragments. For instance, consider the fragment f represented in Fig. 5. It refers to pattern P3. But there also exist two other fragments f_1, f_2 referring to pattern P1 and both are part of f – tasks *manage discount* and *check longevity* represent start nodes of fragments f_1 and f_2. The specification of non-overlapping fragments that actually represent a decision structure concludes the first step as visualized in Fig. 2.

4 Translation of BPMN Decision Logic to DMN

This section discusses the translation of a given process fragment referring to one of the introduced patterns to a DMN model. Before detailing the algorithm, we briefly introduce the Decision Model and Notation (DMN) and provide an example.

4.1 Decision Model and Notation

DMN defines two levels for modeling decision logic, the *decision requirements level* and the *decision logic level*. The first one represents how *decisions* depend on each other and what *input data* is available for the decisions. Therefore, these nodes are connected with each other through *information requirement edges*. A decision may additionally reference the decision logic level where its output is determined through an undirected association. The decision logic level describes the actual decision logic applied to take the decision. Decision logic can be represented in many ways, e.g., by an analytic model or a decision table. In this paper, we utilize decision tables.

Fig. 7 shows an example decision model; *decisions* are rectangles, *input data* are ellipsis, information requirement edges are solid, and the decision table association is dashed. The example is based on the fragment in Fig. 5 from the insurance domain. The decision to be taken refers to the *discount* given to a customer. The corresponding logic is defined in the associated decision table *manage discount table*. The decision cannot be taken directly, since it depends on second level decisions. Information about *Loyalty* and *Longevity* needs to be considered and the results of these decisions are

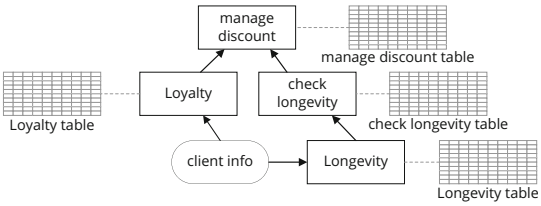


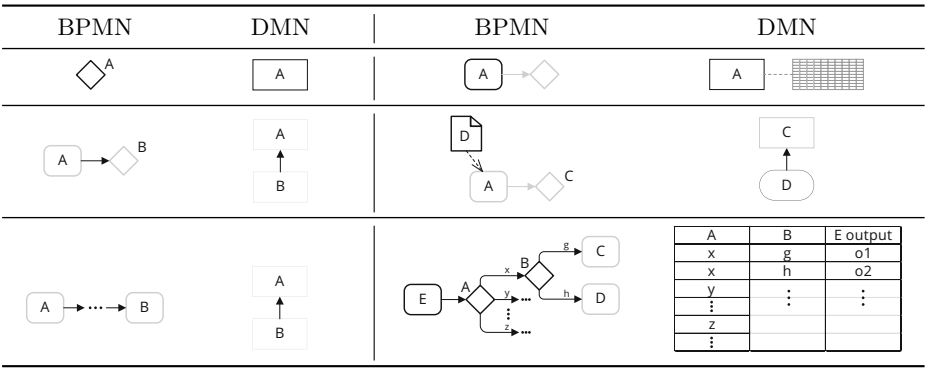
Fig. 7. Example decision model referring to Fig. 5

referenced in the *manage discount table*. *Loyalty* can directly be derived from the input data *client info* while *check longevity* requires the result of *Longevity*.

4.2 Decision Model Extraction Algorithm

Next, we discuss the derivation of decision models from a process fragment satisfying one of the patterns described above. This means that the decision encoded in the fragment is partitioned into a top-level decision connected to sub-decisions with optional input data in DMN. If the decision logic is visible in the process model, we also provide associated decision tables. For this purpose, we devised Table 1 of corresponding model elements that dictates how both the decision requirements level and the decision logic level are constructed.

Table 1. Mapping of BPMN constructs to DMN constructs and the corresponding formalism representations. The black-lined constructs are affected by some mapping while the gray-lined constructs set the context where required.



The left part of row one shows that data-based split gateways are mapped to DMN decision elements because often the data on which the routing is based results from a decision. For example, in Fig. 5, the value of *Loyalty* may need to be inferred from other data such as the number of purchases made so far. Contrarily, note that the value of *Longevity* in Fig. 5 can be observed directly so that in this case the gateway does not need to be mapped to any DMN element. However, since it is hard to differentiate these two situations automatically, the default is to map gateways to independent decision elements. The stakeholders can then decide during post-processing whether or not this is necessary, as described in Section 5.

The right part of row one shows that each BPMN decision task (tasks preceding a gateway) is mapped to a DMN decision element, which is additionally associated with a decision table. Notice that we are able to specify decision tables for decision tasks (right part of row one) but not for gateways (left part). This is because the concrete value of the variable on which the gateway routing is based

usually is set by the task preceding it, and we cannot derive how this is done by only looking at the process model. In case of decision tasks, the situation is different since we can follow each path starting from task A and ending at another task and thereby construct a decision table, as will be explained below for the right part of row three. Because the decision table associated with a decision task will contain the value of the gateway variable, we map the connection of a decision task A and a succeeding gateway B in BPMN to a decision dependency between A and B in DMN. This is shown in the left part of row two.

The right part illustrates how BPMN data nodes are represented in DMN models, if they are available. As just mentioned, we assume that the decision task sets the following gateway's variable. If a data node is connected to this decision task, we assume that the data node is used to arrive at this value. Consequently, in the decision model, the data node is mapped to a DMN input data element providing input to the decision element corresponding to the gateway. The mapping shown in the left part of row three is similar to the one directly above. Decision task B succeeds A without further decision tasks in-between them. Then, in the decision model, decision A uses the output of B as input.

Finally, the right part of row three indicates how decision tables are derived. In general, decision tables consist of rules (represented as rows) having one or more conditions and one conclusion, the columns. Each decision table is associated with one decision element (as can be seen in the right part of the first row). The decision table belonging to decision task E will be made up of all gateways that either follow E or another gateway. The gateway labels are mapped to column headers, and the edge annotations to corresponding column values. There will be as many rows as there are individual paths starting from task E and ending at another task. If any of the gateways on the paths is of type IOR, the decision table's hit policy is set to *multi* since several paths (or rows) can be chosen; otherwise, *single* hit is chosen. The header of the conclusion column is derived from the decision task's label and placeholders are used for its cell values. They are used directly in the refactored process model and can be concretized by the process stakeholders during post-processing.

4.3 Exemplary Decision Model Extraction

This section gives an example for the decision model extraction step described in the previous section using the process fragment satisfying pattern P3 introduced in Section 3. The extraction procedure is best explained with the help of a figure that illustrates the correspondences shown in Table 1 using concrete process and decision models. On the left side of Fig. 8, one can see the process fragment, whereas on the right side the decision model is shown. The latter can be divided into the decision requirements level (top) and the decision logic level (bottom) consisting of decision tables. Also, we inserted arrows to point out the correspondences of the two models' elements. For the sake of clarity, we omitted arrows when the correspondence was already shown by another arrow. For example, arrow 1 shows that the process model's decision task *manage discount* corresponds to the decision element *manage discount* in the decision model

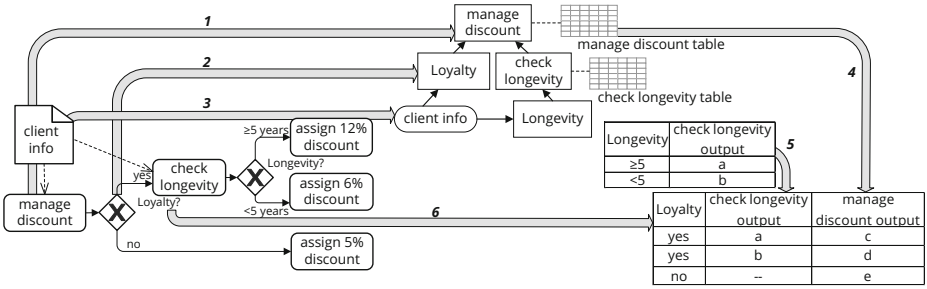


Fig. 8. Exemplary mapping for pattern P3: 1 – from BPMN activity to DMN decision; 2 – from BPMN gateway to DMN decision; 3 – from BPMN data node to DMN input data; 4 – from DMN decision table reference to actual DMN decision table; 5 – from DMN rule conclusions of sub-decision to DMN rule conditions; 6 – from BPMN gateway to DMN rule conditions

(right part of row one in Table 1). Consequently, we did not draw an arrow for the decision task *check longevity*.

Arrow 2 illustrates the left part of row one of Table 1 by mapping the gateway labeled *Loyalty?* to an equivalent decision element. The correspondence between BPMN data nodes and DMN input data elements (right part of row two in Table 1) is demonstrated by arrow 3. Note that the connections between the data node and the tasks in the process model result in connections between the gateway decision elements and the input data in the decision model. Furthermore, the mapping of the left part of row two in Table 1 is demonstrated by the fact that the *manage discount* decision has the *Loyalty* decision as an input requirement. Similarly, corresponding to the left part of row three of that table, since the task *check longevity* succeeds *manage discount*, the DMN decision *manage discount* also requires *check longevity* as an input.

Arrow 4 shows that decisions are connected to decision tables if the decision logic is visible in the process model and arrow 5 shows that the output column of the sub-decision is used as input column of the dependent decision. Arrow 6 visualizes that the headers of the condition columns correspond to the labels of the gateways following the decision task and the cell values equal the edge conditions (cf. right part of row three in Table 1).

4.4 Adaptation of BPMN Models

After extracting the decision logic from a process model to a decision model, the process model needs to be adapted in order to be usable together with the decision model. Basically, the entire decision logic is hidden inside of the first decision task of the pattern. For that purpose, BPMN offers *business rule tasks* that can be linked to decision models and that will output the value of the top-level decision of the decision model. Thus, for the adaptation we transform the task corresponding to this top-level decision to a business rule task. Since this decision

potentially subsumes the decisions corresponding to following decision tasks, these tasks will not be required anymore in the adapted process model. Consequently, we delete each decision task other than the first from the process fragment. Basically, this means that also the gateways succeeding the deleted decision tasks can be removed, such that only the first decision task, the gateway succeeding it and the end nodes of the process fragment are kept. For each end node the gateway has an outgoing edge connected to it and the conditions with which the edges are annotated equal the row conclusions of the top-level decision table.

This situation is illustrated in Fig. 9. It is important to assign the correct conditions to the different edges originating from the split gateway. For example, the end node *assign* 12% *discount* in Fig. 9 is connected to an edge annotated with *c*. This is because in the original process fragment in Fig. 8 the conjunction of the conditions leading from the start node to this end node equals $yes \wedge (\geq 5\ years)$ and the table row representing this conjunction has *c* as its output value.

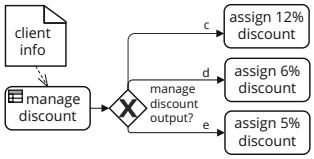
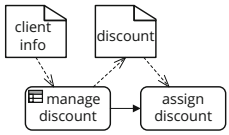


Fig. 9. Refactored process fragment for pattern P3

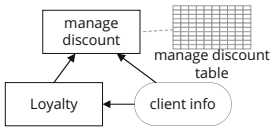
5 Post-Processing

The outcome of the model refactoring step is an adapted process model and corresponding DMN models, one for each decision. On a case basis, these resulting models are not finalized, because, for instance, decisions have been separated which are indeed taken collectively. Thus, we allow to configure the results in two directions: (i) activities taken based on a decision can be combined into single actions such that the branching behavior is replaced and (ii) two connected decisions in the DMN model can be combined. Considering the adapted process fragment shown in Fig. 9, the activities following on each path represent the same action *assign discount* based on some data input representing the actual discount value. In this paper, we require the stakeholder to explicitly specify that such decision shall be reduced with respect to configuration option (i). In future work, we plan to provide this configuration based on, for instance, label analysis. Choice of this option adapts a process fragment as follows: First, the initial split gateway and all succeeding control flow nodes are replaced by a single activity whose label the stakeholder has to specify. Secondly, we add a data node that is written by the decision task and read by the newly added activity. This data node represents the information transferred from the decision to the action taken based on the decision. For the fragment in Fig. 9, the corresponding adapted process fragment is shown in Fig. 10a. The added activity is labeled *assign discount* and the data node is labeled *discount*.

Referring to option (ii) and the DMN model presented in Fig. 8, decisions can be merged. For instance, decisions *Longevity* and *check longevity* can be merged since the first bases on information directly given in the customer information



(a) Decision task without branching behavior.



(b) Adapted DMN model.

Loyalty	Longevity	manage discount output
yes	≥5	c
yes	<5	d
no	--	e

(c) Merged decision table.

Fig. 10. Results of post-processing for the example given in Section 4.3

(time being a customer) and requires no computation. In contrast, the *Loyalty* decision requires computation whether a customer is considered loyal and may not be merged with the *manage discount* decision. Furthermore, decisions *manage discount* and *check longevity* could be merged since both contribute to the decision which actual discount shall be awarded to a customer. Fig. 10b shows the adapted DMN model based on the discussed decision mergers for the outcome of the pattern P3 fragment given in Fig. 9. Merging decisions requires a merge of the corresponding decision tables, i.e., the dependent decision's table is inserted into the higher level decision's table. The resulting table of merging decisions *manage discount* and *check longevity* is shown in Fig. 10c.

After configuring the output models, the stakeholder may adapt the decision tables and the process model a final time. The annotations on the edges originating from a split gateway are intentionally abstract in our approach. The stakeholder may manually add more descriptive annotations by changing the corresponding edge labels or the corresponding rows in the decision task output column.

6 Evaluation

The evaluation is separated into two parts. First, we argue about the benefits of separating process logic and decision logic before we discuss the feasibility of our approach supported by some proof-of-concept implementation. With respect to Parnas [10], system and software design shall follow the concept of "separation of concerns" to utilize specialized concepts and especially one concept per problem resulting in easier maintenance, less complex systems, reusability, flexibility, shortened development time, comprehensibility, and reduced inter-dependencies. Transferring this concept to the process and decision modeling domains, both shall be separated resulting in the same advantages. In detail, a separation provides the following advantages. The complexity of the process model is reduced while, at the same time, precision, readability, and maintainability (of both the decision and the process model) get improved. Additionally, business logic and decision logic can be changed individually resulting in reduced changing costs and changing times, e.g., compliance experts can tune specific decision points without changing the process structure. Process models are considered stable and only to be adapted if the business changes while decision models are considered dynamic to react fast

and flexible on temporary situations. Furthermore, separating the decision logic from process model logic allows reusability of the decision model in multiple decisions occurring in the same as well as different process models.

While separation of both worlds is easy for newly modeled processes, the existing ones need to be kept usable as well. Otherwise, the migration overhead is too large and organizations retain their BPMN misuse. For utilizing the advancements of simplicity, easy maintainability, high precision, and automatic analyzability, the original process model gets adapted to replace the decision logic fragment with a reference to the DMN model after its creation. Based on the assumption that the concept of separation of concerns can be transferred to the process and decision domains, the usefulness of the presented approach for stakeholders is directly given.

The ultimate goal with respect to separation of concerns of process and decision logic is to remove the decision logic entirely from the process model. This results in a transformation of a decision – simple ones as represented by pattern P1 as well as most complex decision structures – into a single activity as shown in Fig. 10a.

To reason about applicability of our approach introduced in this paper, we generically compare the decision logic of both the original and the adapted process model fragments. As mentioned in Section 4, decision tables consist of rules made up of conjunctions of conditions and one conclusion. The conjuncts of a rule are equal to the conditions annotated to the edges originating from the split gateways that are on an individual path from start to end node of a fragment, while the conclusion is a placeholder to be used as an edge condition in the adapted fragment. Consequently, both the original and the adapted fragment together with the extracted decision tables represent the same decision logic. This directly shows that the same end node is reached in both process fragments.

We also implemented the introduced approach. We utilized an open source platform for research on process model repositories [4] that is based on the pipe and filter technique principle. Our implementation extending the platform with further modules, their documentation, and some example process models are available at *http://bpt.hpi.uni-potsdam.de/Public/BpmnDmn*. We used this implementation to validate the impact of our approach to the process model repositories of our project partners from the domains of insurance, banking, healthcare, energy, and information technology.

In total, we received 956 process models from them with 566 being syntactically correct. Applying our implementation on these syntactically correct industry process models reveals that pattern P1 occurs in 59%, pattern P2 in 16%, and pattern P3 in 32% of all process models as visualized in Fig. 11. In total, we observed 680 occurrences of pattern P1 fragments, 113 occurrences of pattern P2 fragments, and 362 occurrences of pattern P3

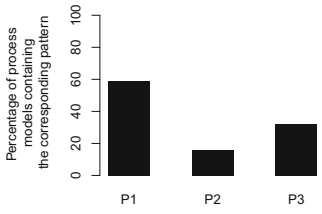


Fig. 11. Pattern occurrence in real world process models

fragments our set of 566 models. These numbers show that the identified patterns are frequently used in practice and thus, we already provide high impact by handling them.

7 Related Work

It is not considered good practice to model the detailed decision paths in the business process model. Thus, there is a demand for finding a good integration between decision and process modeling both in industry and academia. In order to deal with the changes which can arise from run-time contextual changes or the change of user requirements and preferences, different approaches on decision services modeling are used and proposed.

Similarly to our point of view, separation of decision from process logic is discussed in [3,6,14]. [6] presents a tool chain for creation of both models relying on concrete infrastructure and business rules. In contrast, we introduce generic means based on two standards of the OMG – BPMN and DMN. [14] and [3] also utilize these standards but do not provide the next step – as [6] also does not – that is important for practical usage: migration of existing process models into the new separated structure.

Apart from the DMN approach, there are other outlooks dedicated to model the separation of decision-making from application process logic. For instance, in [16], the authors come up with an approach which supports an asynchronous interaction such that the decision service can notify the process about new changes at any time and not only at predefined decision points. [7] presents a decision ontology for supporting decision-making in information systems. Another decision ontology is proposed in [1] as a "domain-specific modeling method that is integrated with an existing enterprise modeling method for describing and communicating decision processes". There are some industrial solutions for separation of decision from process logic, e.g., SAP Decision Service Management [13]. In contrast to our approach, these do not use standards; especially not those that have been designed to solve the advancement of logic separation. Here, BPMN provides rule-tasks that reference a DMN model allowing proper integration of both worlds. Thus, we decided to consider the DMN standard for exploiting the decision logic alongside BPMN for exploiting the process logic.

Another direction of extracting decision logic from process models is "decision mining" [11,12]. Though decision mining helps with the analysis of the dependencies within a process model, the advantage of our approach is that it is purely model-based and does not require additional information such as execution logs. Further, the existing approaches do not cover automatic refactoring of process models.

In [5], the authors state that declarative modeling approaches lead to more design- and run-time flexibility, better compliance guarantees, and higher expressibility. Since decision models are declarative in nature [14], combining them with business process models will provide these benefits as well and additionally preserves the benefits from the concept of separation of concerns.

8 Conclusion

In this paper, we elaborated on the advantages to separate process and decision logic resulting in simpler models with easy maintainability, high precision, and automatic analyzability. The separation especially fosters the differentiation of stable process models to be changed if the business model changes and dynamic decision models allowing flexible configuration of the currently applied business models. Since organizations long misused BPMN as decision modeling languages although it is not meant to capture these aspects, organizations require migration capabilities from misused, spaghetti like process models to a separation that we build of an adapted BPMN model and a DMN model. We provide a semi-automatic approach that allows identification of decision logic in process models, derivation of corresponding DMN models, adaptation of the original process model by replacing the decision logic accordingly, and final configuration of the result during post-processing. The identification is pattern-based derived from an intensive analysis of 956 real world process models provided by our project partners. We implemented this semi-automatic approach and provided statistical insights about pattern utilization in the industry process models. Since AND gateways do not influence decisions (neither AND forks nor merges) and since explicitly considering them would significantly increase the complexity of our approach and its formalization, we disregard AND gateways in this paper. Although not stated explicitly in the patterns, we also support loops like *WHILE x DO y*, since in these cases, the same decision is taken multiple times with varying input data values until the looping condition evaluates to false.

In future work, we will analyze further process model collections and reduce the assumption of control flow decision structures to identify more patterns and to provide a complete overview about decision logic modeling in process models. We also aim on pattern generalization. The mapping will be adjusted accordingly. Furthermore, we extend the configuration capabilities by, e.g., including label analysis. Finally, we plan to publish best practice guidelines on how to model processes and decisions separately.

Acknowledgments. The research leading to these results has been partly funded by DFG under grant agreement WE 1930/8-1. We thank Kristina Kirsten, Tobias Rohloff, and Thomas Zwerg for the support in analyzing the industry process repositories.

References

1. Bock, A., Kattenstroth, H., Overbeek, S.: Towards a modeling method for supporting the management of organizational decision processes. In: Modellierung, vol. 225, pp. 49–64. Gesellschaft für Informatik (2014)
2. Catalkaya, S., Knuplesch, D., Chiao, C., Reichert, M.: Enriching business process models with decision rules. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013 Workshops. LNBIP, vol. 171, pp. 198–211. Springer, Heidelberg (2014)
3. Debevoise, T., Taylor, J.: The MicroGuide to Process Modeling and Decision in BPMN/DMN. CreateSpace Independent Publishing Platform (2014)

4. Eid-Sabbagh, R.-H., Kunze, M., Meyer, A., Weske, M.: A platform for research on process model collections. In: Mendling, J., Weidlich, M. (eds.) BPMN 2012. LNBIP, vol. 125, pp. 8–22. Springer, Heidelberg (2012)
5. Goedertier, S., Vanthienen, J., Caron, F.: Declarative business process modelling: principles and modelling languages. Enterprise Information Systems **9**(2), 161–185 (2015)
6. Kluza, K., Kaczor, K., Nalepa, G.J.: Integration of business processes with visual decision modeling. Presentation of the HaDEs toolchain. In: Fournier, F., Mendling, J. (eds.) BPM 2014 Workshops. LNBIP, vol. 202, pp. 504–515. Springer, Heidelberg (2015)
7. Kornyshova, E., Deneckère, R.: Decision-making ontology for information system engineering. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 104–117. Springer, Heidelberg (2010)
8. OMG: Business Process Model and Notation (BPMN), Version 2.0 (January 2011)
9. OMG: Decision Model and Notation (February 2014)
10. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM **15**(12), 1053–1058 (1972)
11. Petrusel, R.: Using markov decision process for recommendations based on aggregated decision data models. In: Abramowicz, W. (ed.) BIS 2013. LNBIP, vol. 157, pp. 125–137. Springer, Heidelberg (2013)
12. Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 420–425. Springer, Heidelberg (2006)
13. SAP: SAP Decision Service Management. http://scn.sap.com/docs/DOC-29158 (accessed: November 13, 2014)
14. Von Halle, B., Goldberg, L.: The Decision Model: A Business Logic Framework Linking Business and Technology. Taylor and Francis Group (2010)
15. Weske, M.: Business Process Management: Concepts, Languages, Architectures, 2nd edn. Springer, Heidelberg (2012)
16. Zarghami, A., Sapkota, B., Eslami, M.Z., van Sinderen, M.: Decision as a service: separating decision-making from application process logic. In: EDOC, pp. 103–112. IEEE (2012)